ЗАДАНИЕ

Написать распределенное клиент-серверное приложение, используя указанную технологию распределенной объектной обработки. Приложение реализует игру «Змейка» со следующими правилами:

– в игре присутствует один игрок-пользователь;

– пользователь управляет движущимся объектом (змейкой);

– змейка выполняет задания серверного приложения (например, пересечь заданную точку на экране, выполнить заданную траекторию и т.д.);

– при успешном выполнении задания игрок получает некоторое вознаграждение (например, змейка увеличивается, игрок получает баллы и т.д.);

– игра прекращается при нарушении которых ограничений игры (например, выход за границу игрового поля, пересечение змейкой самой себя, и т.д.).

Вариант 3. Технология servlets (JSP, JSF).

РЕФЕРАТ

Отчет по курсовой работе: 32 с., 12 рисунков, 7 источников, 1 приложение.

ОБЪЕКТНО РАСПРЕДЕЛЁННАЯ ОБРАБОТКА, ВЕБ-ПРИЛОЖЕНИЕ, КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, ЗМЕЙКА, ЯЗЫК UML, СЕРВЛЕТЫ, JAVA, JSP.

В ходе выполнения курсовой работы было разработано клиент-серверное приложение, представляющее собой реализацию компьютерной игры «Змейка». Серверное приложение определяет правила игры, данные игры, порядок и форму обработки запросов от клиентской части, порядок и форму отклика в клиентскую часть приложения. Клиентское приложение формирует запрос в виде, требуемом серверным приложением, а также интерпретирует полученный ответ с учетом эргономичности и комфорта для пользователя.

Приложение написано на языке Java с использованием среды разработки Intellij IDEA.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 5

1 Описание и анализ предметной области 6

1.1 Описание предметной области 6

1.2 Описание используемых технологий 7

1.2.1 Сервлеты 7

1.2.2 Java Server Pages (JSP) 7

2 Проектирование системы 9

2.1 Структурная схема системы 9

2.2 Разработка прототипа интерфейса пользователя системы 10

2.3 Разработка информационно-логического проекта системы 14

2.3.1 Язык UML 14

2.3.2 Диаграмма вариантов использования 14

2.3.3 Диаграмма состояний 15

2.3.4 Диаграмма деятельности 18

3 Реализация системы 20

3.1 Описание пользовательского интерфейса 20

ЗАКЛЮЧЕНИЕ 23

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 24

ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД 25

ВВЕДЕНИЕ

Целью данной курсовой работы является создание клиент-серверного приложения, закрепление навыков работы с технологиями для решения поставленной задачи распределённой обработки, определение и формализация требований к системе, накладываемых реализуемой технологией обработки.

В ходе выполнения курсовой работы необходимо разработать программную систему распределённой обработки данных, реализующую предложенное задание с помощью одной из предложенных технологий распределенной обработки.

В первой главе настоящей записки представлено описание и анализ предметной области, описание технологий, используемых в ходе курсового проектирования. Во второй главе представлено описание проекта разрабатываемой системы на языке UML. В третьей главе описана реализация программной системы распределённой обработки данных.

1 Описание и анализ предметной области

1.1 Описание предметной области

Согласно одному из многочисленных определений, распределенная система – это система, состоящая из набора двух или более независимых узлов, которые координируют свою работу посредством синхронного или асинхронного обмена сообщениями.

В настоящей курсовой работе для создания распределённой системы применено функциональное разделение: на клиентскую и серверную часть. Таким образом, различные узлы системы будут выполнять разные задачи.

Snake («Питон», «Удав», «Змейка» и др.) — компьютерная игра, возникшая в середине или в конце 1970-х [1].

Игрок управляет длинным, тонким существом, напоминающим змею, которое ползает по плоскости (как правило, ограниченной стенками), собирая еду (или другие предметы), избегая столкновения с собственным хвостом и краями игрового поля. В некоторых вариантах на поле присутствуют дополнительные препятствия. Каждый раз, когда змея съедает кусок пищи, она становится длиннее, что постепенно усложняет игру.

Игрок управляет направлением движения головы змеи (обычно 4 направления: вверх, вниз, влево, вправо), а хвост змеи движется следом. Игрок не может остановить движение змеи. На рисунке 1 представлен пример игры «Змейка».

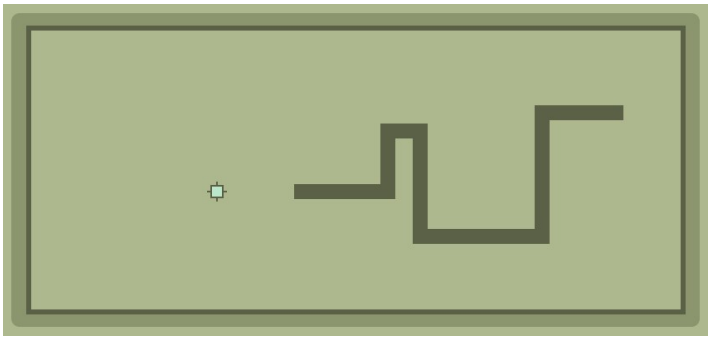

Рисунок 1 – Пример игры «Змейка»

1.2 Описание используемых технологий

Согласно варианту курсовой работы, для реализации распределённой системы необходимо использовать сервлеты и JSP (или JSF).

1.2.1 Сервлеты

Сервлет является интерфейсом Java, реализация которого расширяет функциональные возможности сервера. Сервлет взаимодействует с клиентами посредством принципа запрос-ответ [2].

Для обеспечения возможности работы с сервлетами HTTP-сервер должен содержать в себе контейнер сервлетов, например, Tomcat.

Жизненный цикл сервлета:

– Если экземпляр сервлета на существует, контейнер загружает класс сервлета, создает его экземпляр, вызывает метод init().

– При получении запроса вызывается метод service(), которому передаются объекты запроса и отклика.

– Если контейнеру требуется удалить объект сервлета, то вызывается метод destroy().

1.2.2 Java Server Pages (JSP)

JSP – это технология, которая упрощает создание веб-страниц с динамически изменяющимся содержимым [3].

Технология JSP позволяет сочетать статические элементы (например, html-теги) и динамические JSP элементы (например, исполняемый код на языке Java). Такой подход позволяет гибко и быстро создавать веб-страницы.

JSP-элементы представляют собой набор синтаксических выражений, которые упрощают разработку веб-страницы. Ярким примером JSP-элемента являются вставки java-кода или же специальные теги, которые не входят в стандартный набор HTML тегов.

Иногда использование стандартных способов для ряда операций, например, для вывода на страницу элементов из списка и т.д., может быть несколько громоздким. Чтобы упростить встраивание кода java в JSP была разработана специальная библиотека – JSTL. JSTL (JSP Standard Tag Library) предоставляет теги для базовых задач JSP (цикл, условные выражения и т.д.).

При компиляции приложения все JSP-страницы преобразуются в сервлеты, прослушивающие подключения на адрес, которым является название файла, содержащего JSP-страницу. Таким образом, при написании JSP-страницы можно пользоваться всеми возможностями сервлетов.

2 Проектирование системы

2.1 Структурная схема системы

Система – множество элементов, находящихся в отношениях и связях друг с другом, которое образует определённую целостность, единство [4].

Система должна представлять собой совокупность элементов (объектов, субъектов), находящихся между собой в определённой зависимости и составляющих некоторое единство (целостность), направленное на достижение определённой цели. Система может являться элементом другой системы более высокого порядка (надсистема) и включать в себя системы более низкого порядка (подсистемы). То есть систему можно рассматривать как набор подсистем, организованных для достижения определённой цели и описанных с помощью набора моделей (возможно, с различных точек зрения), а подсистему – как группу элементов, часть которых составляет спецификацию поведения, представленного другими ее составляющими.

На рисунке 2 приведена структурная схема разрабатываемой системы. Система разделена на две части (клиентская и серверная части, которые обмениваются информацией через HTTP-протокол). В состав разрабатываемой системы клиентской части входят следующие подсистемы:

1) подсистема взаимодействия с серверной частью, отвечающая за передачу на сервер информации об итогах игры (длина змейки);
2) подсистема обработки действий пользователя, отвечающая за обеспечение реакции на действия, совершаемые пользователем, и за передачу данных о совершенных действиях в подсистему взаимодействия с серверной частью;
3) подсистема визуализации, отвечающая за отображение игрового поля и других объектов.

В состав серверной части системы входят следующие подсистемы:

1) подсистема взаимодействия с клиентской частью, отвечающая за передачу игровой модели пользователю и информации об итогах игры (вывод результата игры);

2) игровая модель, которая отвечает за игровой процесс.

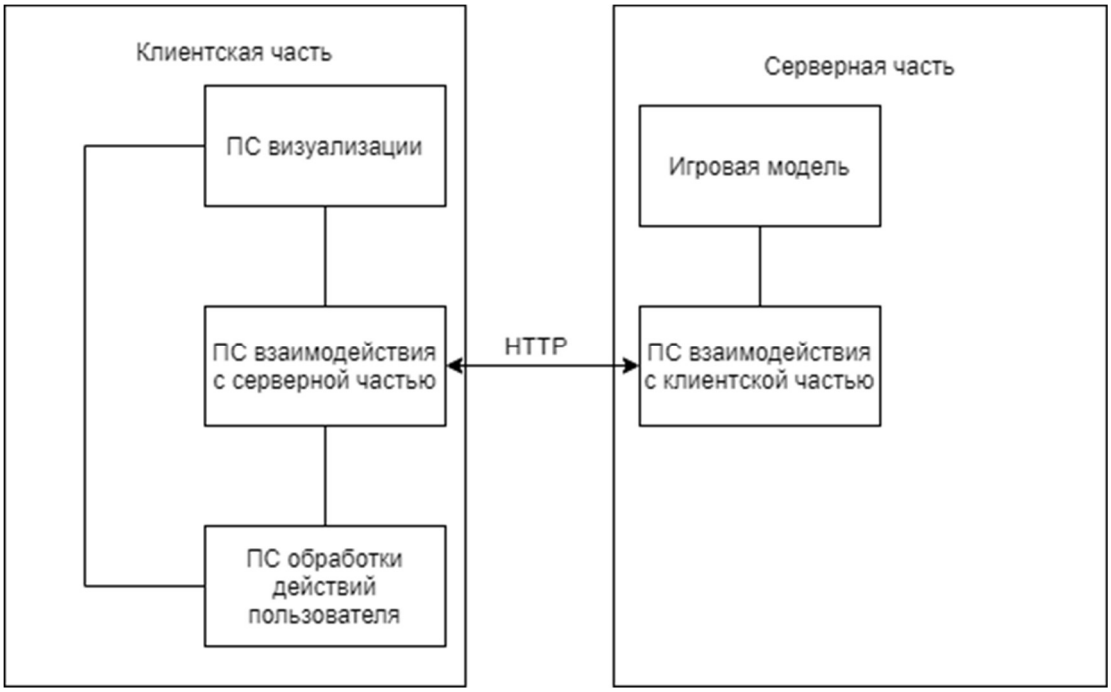

Рисунок 2 – Структурная схема системы

2.2 Разработка прототипа интерфейса пользователя системы

Интерфейс пользователя – совокупность средств, при помощи которых пользователь общается с различными устройствами, чаще всего с компьютером или бытовой техникой, либо иным сложным инструментарием (системой) [5].

Интерфейс пользователя компьютерного приложения включает:

– средства отображения информации, отображаемую информацию, форматы и коды;

– командные режимы, язык «пользователь – интерфейс»;

– устройства и технологии ввода данных;

– диалоги, взаимодействие и транзакции между пользователем и компьютером, обратную связь с пользователем;

– поддержку принятия решений в конкретной предметной области;

– порядок использования программы и документацию на неё.

Процесс создания интерфейса начинается с определения целей проекта, а также внутренних и внешние обстоятельств, которые вы должны принять во внимание. Для того чтобы правильно расставить приоритеты, необходимо учитывать:

– опыт работы пользователей с компьютером, типовые ситуации использования;

– какая информация необходима и когда, какие результаты должны быть получены;

– технологию разработки и платформа, на которой будут работать пользователи.

На рисунке 3 представлен прототип пользовательского интерфейса для начальной страницы приложения.

На рисунке 4 представлен прототип справочной информации, который будет появляться при нажатии на кнопку «Справка» на начальной странице приложения.

После того как пользователь начнёт игру, отобразится страница с игровым полем. Прототип пользовательского интерфейса игрового процесса представлен на рисунке 5.

При завершении игры отображается страница с данным сообщением и предложением пользователю начать новую игру. Прототип пользовательского интерфейса для страницы завершения игры представлен на рисунке 6.

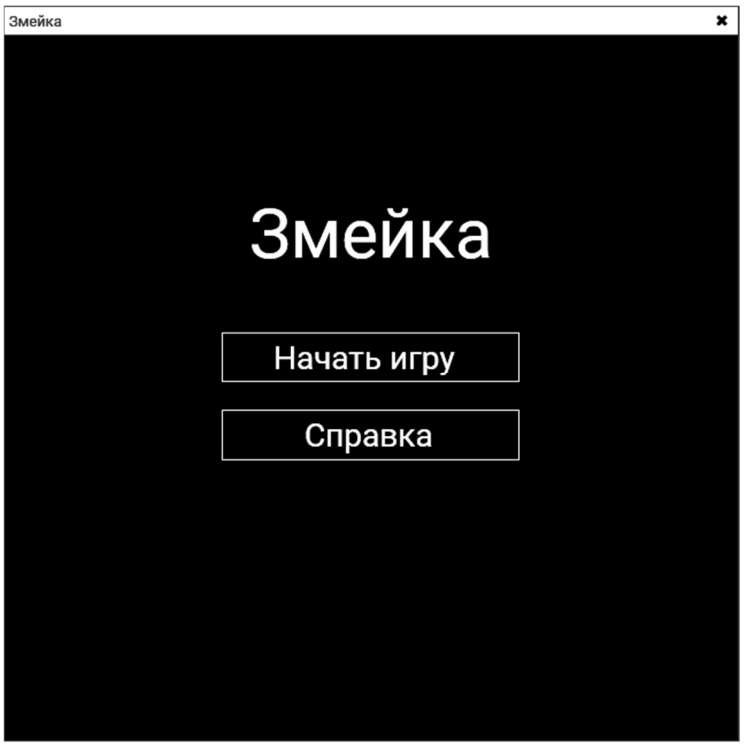

Рисунок 3 – Прототип начальной страницы приложения

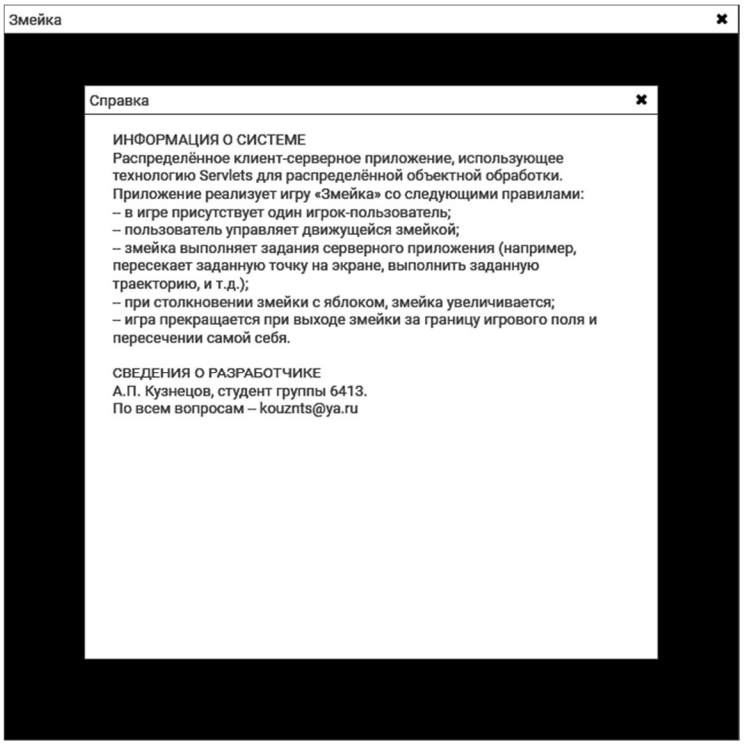

Рисунок 4 – Прототип окна со справочной информацией

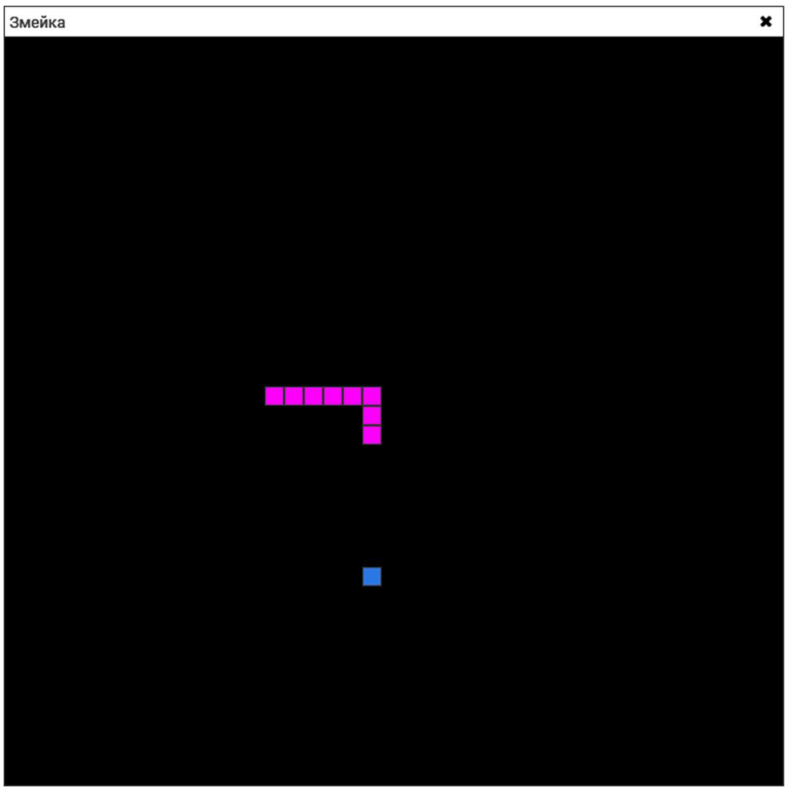

Рисунок 5 – Прототип страницы с игровым полем

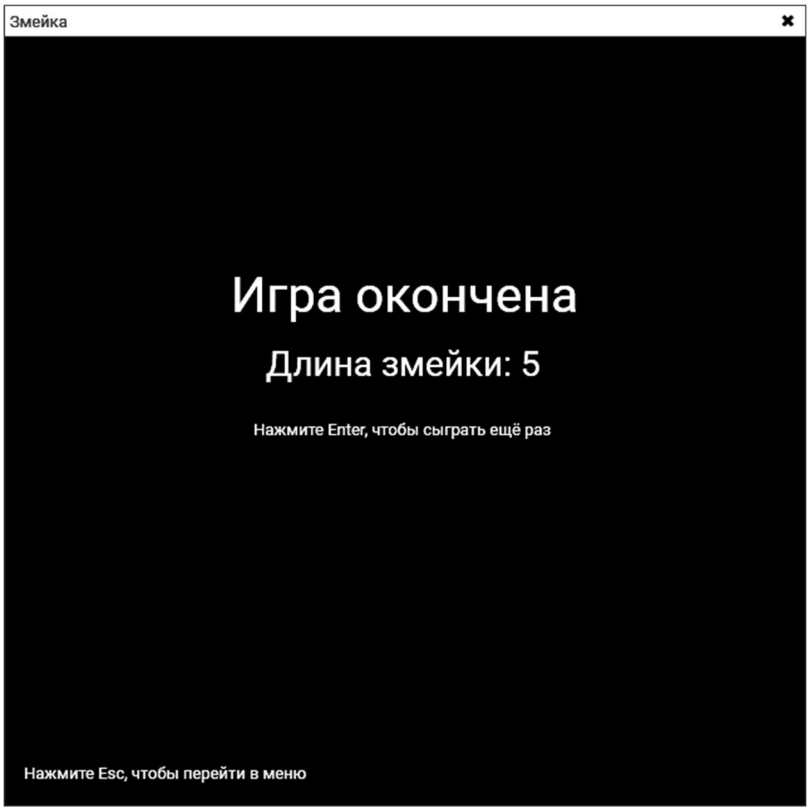

Рисунок 6 – Прототип страницы завершения игры

2.3 Разработка информационно-логического проекта системы

2.3.1 Язык UML

Унифицированный язык моделирования (Unified Modeling Language – UML) – это стандартный инструмент для разработки «чертежей» программного обеспечения. Его можно использовать для визуализации, спецификации, конструирования и документирования артефактов программных систем. Язык UML предоставляет стандартный способ написания проектной документации на системы, включая концептуальные аспекты, такие как бизнес-процессы и функции системы, а также конкретные аспекты, такие как выражения языков программирования, схемы баз данных и повторно используемые компоненты программного обеспечения.

2.3.2 Диаграмма вариантов использования

Диаграмма вариантов использования представляет собой наиболее общую концептуальную модель сложной системы, которая является исходной для построения всех остальных диаграмм [6]. На этой диаграмме изображаются отношения между актёрами и вариантами использования, а также описывается функциональное назначение системы.

Для дальнейшего понимания сути диаграммы вариантов использования или use case diagram введем следующую терминологию.

Актёр (actor) – согласованное множество ролей, которые играют внешние сущности по отношению к вариантам использования при взаимодействии с ними (это может быть любой объект, субъект или система, взаимодействующая с моделируемой бизнес-системой извне, т.е. человек, техническое устройство, программа и т.п.).

Вариант использования – внешняя спецификация последовательности действий, которые система или другая сущность могут выполнять в процессе взаимодействия с актёрами (он определяет набор действий, совершаемый системой при диалоге с актёром).

Цель спецификации варианта использования заключается в том, чтобы зафиксировать некоторый аспект или фрагмент поведения проектируемой системы без указания особенностей реализации данной функциональности.

На рисунке 7 представлена диаграмма вариантов использования для создаваемого приложения.

2.3.3 Диаграмма состояний

Для моделирования поведения на логическом уровне в языке UML могут использоваться сразу несколько канонических диаграмм: состояний, деятельности, последовательности и кооперации, каждая из которых фиксирует внимание на отдельном аспекте функционирования системы. В отличие от других диаграмм диаграмма состояний описывает процесс изменения состояний только одного экземпляра определенного класса, т. е. моделирует все возможные изменения в состоянии конкретного объекта.

Главное предназначение этой диаграммы – описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла. Диаграмма состояний представляет динамическое поведение сущностей, на основе спецификации их реакции на восприятие некоторых событий [4].

На рисунке 8 изображена диаграмма состояний системы.

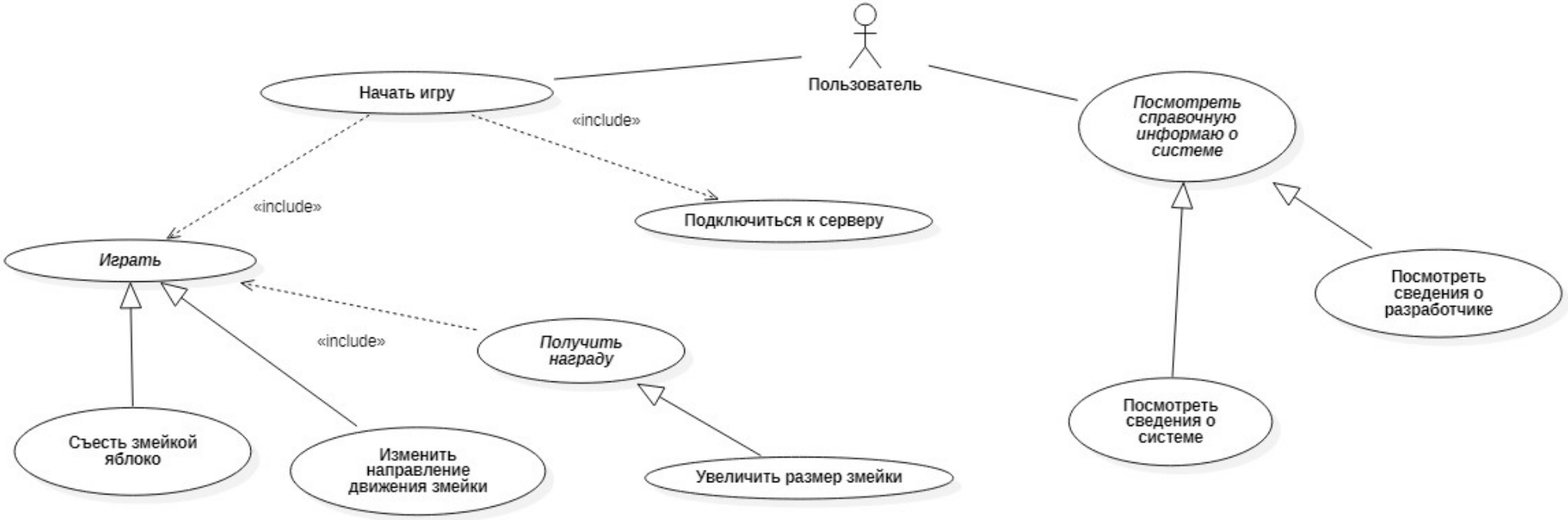

Рисунок 7 – Диаграмма вариантов использования

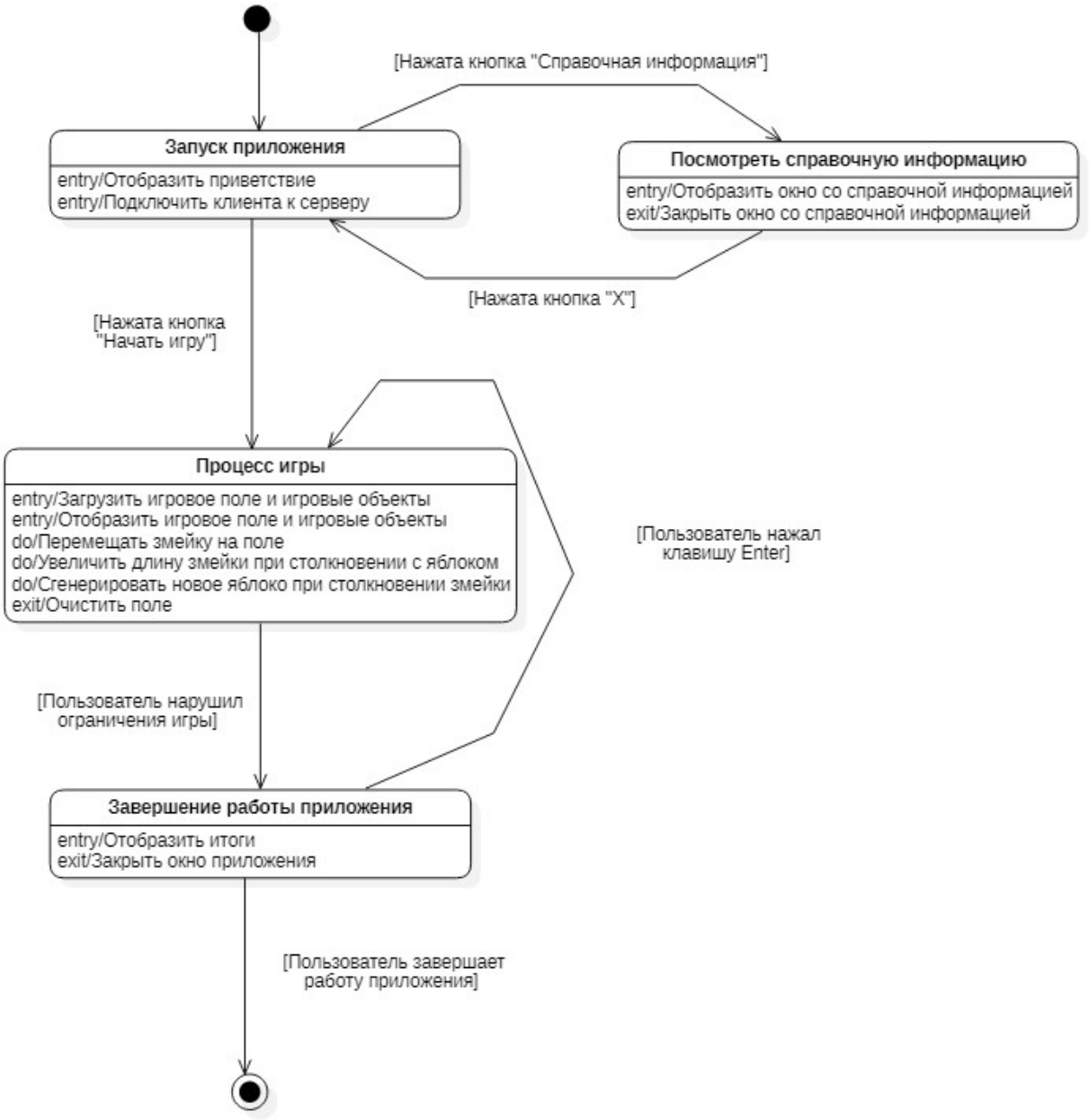

Рисунок 8 – Диаграмма состояний системы

При открытии клиентской части приложения происходит подключение к серверу. Сервер передаёт клиенту модель игры. Клиент может посмотреть справочную информации или начать игровой процесс. После выбора начать игровой процесс, клиент получает игровое поле и объекты, после чего происходит отрисовка полученного поля и объектов. В начале игрового процесса змейка начинает движение в одном направлении до тех пор, пока она не случиться коллизия с краем игрового поля или пока клиент не изменит направление движении змейки. Пользователь может изменить направление

змейки с помощью стрелок. Когда змейка съедает яблоко на игровом поле, то её длинна увеличивается и происходит генерация новой координаты яблока. Координаты яблока генерируются случайным образом. Игра заканчивается, когда случает коллизия змейки с краем поля или самой собой. При завершении игры происходит передача данных на сервер (данные содержат информацию о длине змейки). В ответ сервер присылает страницу, на которой отображается статистика игровой сессии. Пользователь может начать игровой процесс заново или же вернуться на начальную страницу.

2.3.4 Диаграмма деятельности

Диаграмма деятельности – это диаграмма, применяемая в UML для моделирования динамических аспектов систем [7]. По сути, диаграмма деятельности представляет собой блок-схему, которая показывает, как поток управления переходит от одной деятельности к другой.

Моделирование динамических аспектов систем при помощи диаграмм деятельности большей частью подразумевает моделирование последовательных шагов вычислительного процесса. Диаграммы деятельности могут использоваться отдельно для визуализации, специфицирования, конструирования и документирования динамики сообщества объектов либо для моделирования потока управления в операции. Деятельность – это структурированное описание текущего поведения [7].

На рисунке 9 приведена диаграмма деятельности для системы. На данной диаграмме отображен игровой процесс.

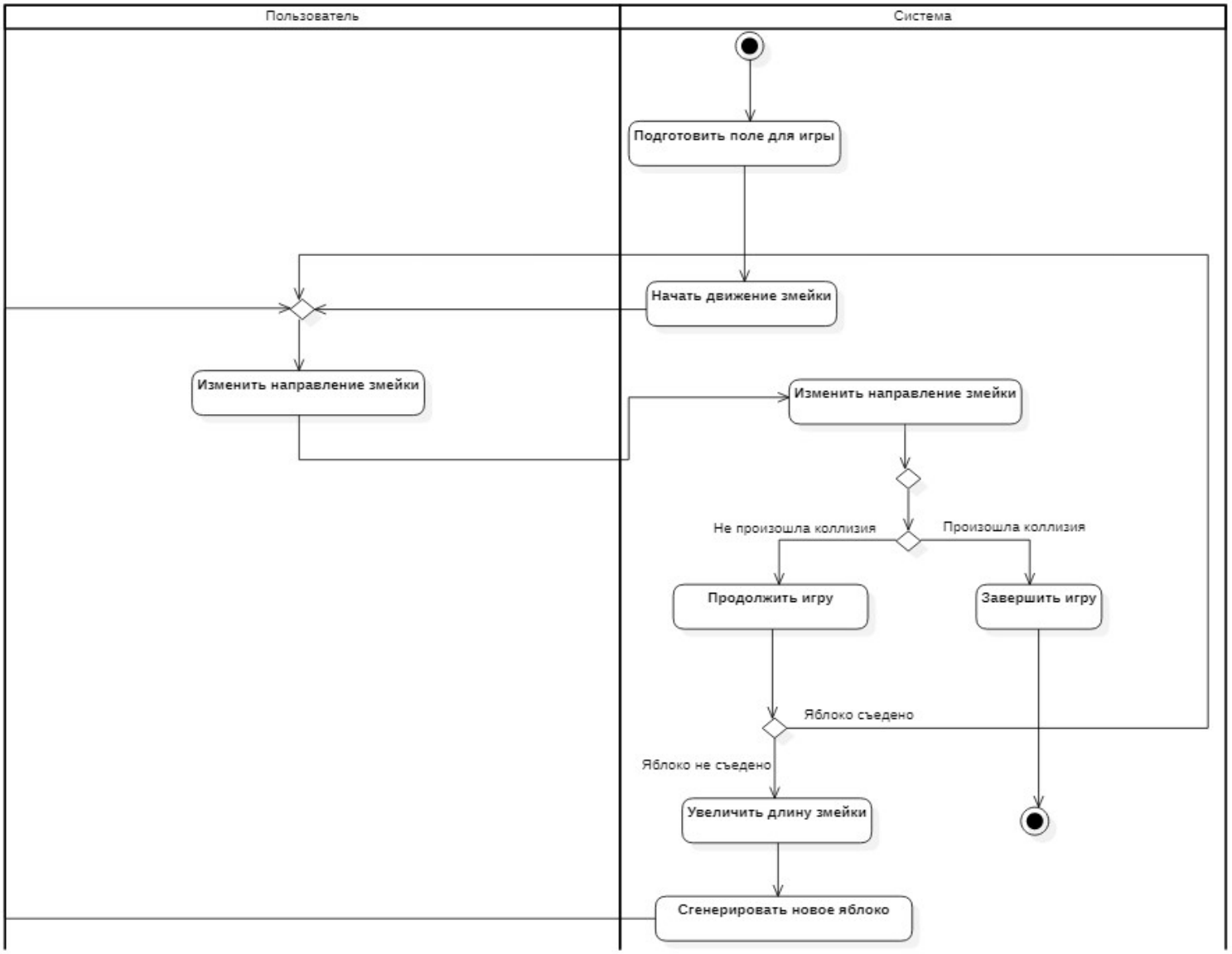

Рисунок 9 – Диаграмма деятельности

3 Реализация системы

3.1 Описание пользовательского интерфейса

При запуске приложения пользователь видит на экране начальную страницу, на которой он может посмотреть справочную информацию или начать игру. Изображение содержательной части данной веб-страницы представлено на рисунке 10.

Рисунок 10 – Начальный экран приложения

При нажатии на кнопку «Начать игру» будет начат процесс игры. При этом будет отражено поле, на котором будет передвигаться змейка, и яблоко, которое генерируется случайным образом. Процесс игры представлен на рисунке 11.

Каждый шаг змейки обрабатывается на стороне клиента. Потом, когда яблоко съедается, происходит генерация новых координат яблока. Данные координат генерируются случайным образом на стороне клиента.

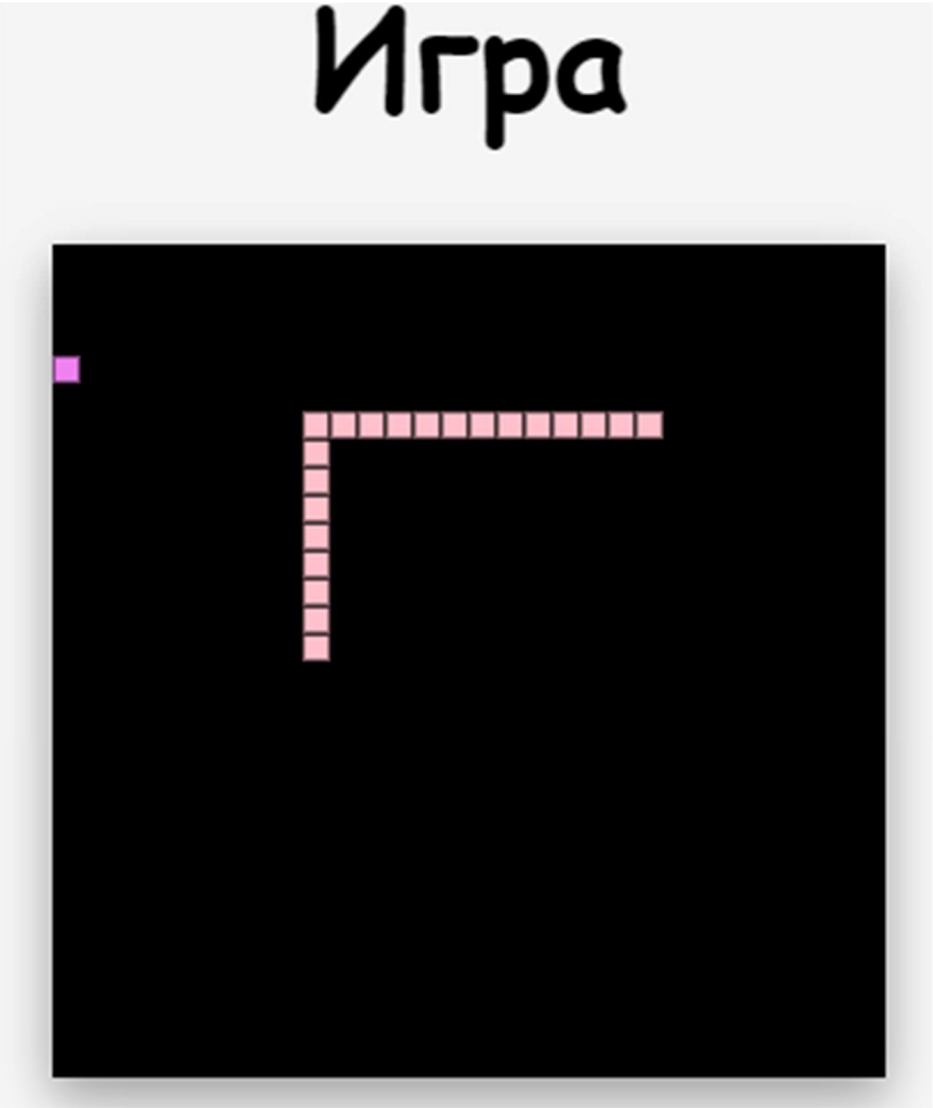

Рисунок 11 –Игровой процесс

Когда происходит коллизия, то серверу приходят соответствующие данные и игра завершается. Осуществляется переход к следующей странице, на которой указаны результаты статистики. На рисунке 12 представлена данная страница.

Клиент может начать игру заново, нажав на клавишу «Enter» или может вернуться на начальную страницу приложения, нажав стрелку влево.

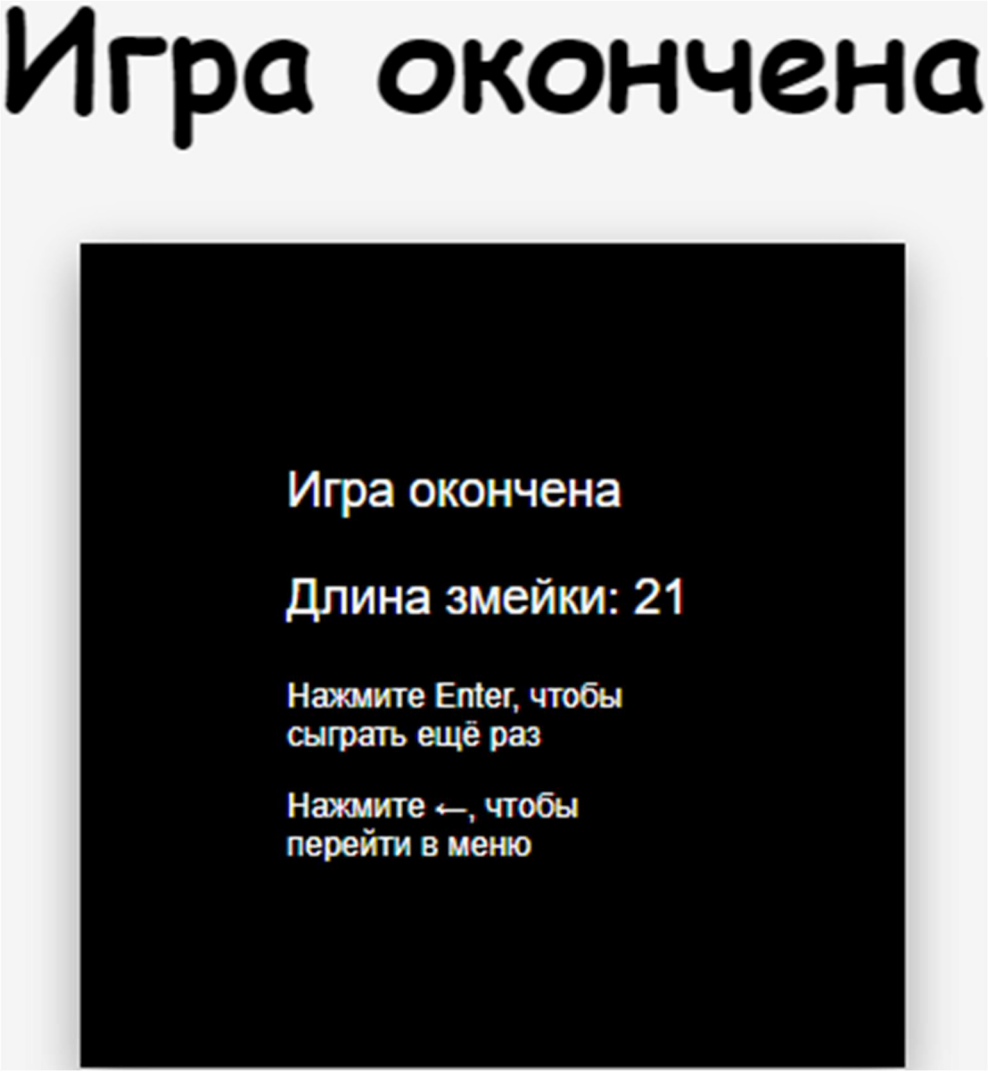

Рисунок 12 – Результат статистики

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсовой работы было разработано распределенное приложение с клиент-серверной архитектурой, которое позволяет пользователю играть в игру «Змейка».

В первой главе настоящей записки представлено описание и анализ предметной области, описание технологий, используемых в ходе курсового проектирования.

Во второй главе представлено описание проекта разрабатываемой системы, в том числе на языке UML. Также представлены прототипы оконного интерфейса пользователя.

В третьей главе описана реализация программной системы распределенной обработки данных, приведены скриншоты работы программы при различных действиях пользователя.

В приложении А приведен исходный код разработанной системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Змейка [Электронный ресурс] // Википедия – Электрон. дан. – Б. м., 2020. – URL: https://ru.wikipedia.org/wiki/Snake_(игра) (дата обращения: 23.04.2020).

2 Сервлет (Java) [Электронный ресурс] // Википедия – Электрон. дан. – Б. м., 2020. – URL: https://ru.wikipedia.org/wiki/Сервлет_(Java) (дата обращения: 23.04.2020).

3 Java Server Pages [Электронный ресурс] // Википедия – Электрон. дан. – Б. м., 2020. – URL: https://ru.wikipedia.org/wiki/JavaServer_Pages (дата обращения: 23.04.2020).

4 Буч Г., Рамбо Д., Якобсон А. Язык UML. Руководство пользователя. Изд. 2-е. М.: ДМК Пресс, 2006. С. 21.

5 Пользовательский интерфейс [Электронный ресурс] // Словари и энциклопедии: [сайт]. – URL: https://dic.academic.ru/dic.nsf/ruwiki/1100993 (дата обращения: 24.04.2020).

6 Зеленко Л.С. Методические указания к лабораторному практикуму по дисциплине «Программная инженерия». Самара: СГАУ, 2012. 67 с.

7 Диаграмма деятельности [Электронный ресурс] // Язык UML Руководство пользователя: официальный интернет-сайт. – Электрон. дан. – Б. м., 2020.– URL: http://bourabai.ru/dbt/uml/ch19.htm (дата обращения: 15.04.2020).

game.jsp

```
<%@ page contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8" %>
<!DOCTYPE html PUBLIC
"-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd" >
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
   <meta http-equiv="Content-Type"
         content="text/html; charset=ISO-8859-1">
   <style type="">
      <%@include file="/style.css" %>
   </style>
   <title>Игра</title>
</head>
<body>
<h1>Игра</h1>
<canvas id="game-canvas" width="300" height="300"></canvas>
<script type="">
   const CANVAS_BORDER_COLOR = 'black';
   const CANVAS_BACKGROUND = 'black';
   const SNAKE_COLOR = 'pink';
   const SNAKE_BORDER_COLOR = 'white';
   const APPLE_COLOR = 'violet';
   const APPLE_BORDER_COLOR = 'white';
```

```
let snake = [
    {x: 150, y: 150},
    {x: 140, y: 150},
    {x: 130, y: 150},
    {x: 120, y: 150},
    {x: 110, y: 150}
];
let score = snake.length;
let xdir = 10;
let ydir = 0;
const canvas = document.getElementById("game-canvas");
const gameCanvas = canvas.getContext("2d");
gameCanvas.fillStyle = CANVAS_BACKGROUND;
gameCanvas.strokestyle = CANVAS_BORDER_COLOR;
gameCanvas.fillRect(0, 0, canvas.width, canvas.height);
gameCanvas.strokeRect(0, 0, canvas.width, canvas.height);
startGame();
createApple();
document.addEventListener("keydown", changeSnakeDirection);
function startGame() {
   if (isEndGame()) {
       let form = document.createElement('form');
       form.action = '${pageContext.request.contextPath}/gameover';
       form.method = 'POST';
       form.innerHTML = '<input id="score" type="hidden" name="score">';
       document.body.append(form);
       document.getElementById("score").value = snake.length;
```

```
        form.submit();
        return;
    }
    setTimeout(function onTick() {
        clearGameCanvas();
        drawApple();
        moveSnake();
        drawSnake();
        startGame();
    }, 100);
}
function isEndGame() {
    return isSnakeCollision() || isWallCollision();
}
function isSnakeCollision() {
    for (let i = 4; i < snake.length; i++) {
        const isCollision =
            snake[i].x === snake[0].x &&
            snake[i].y === snake[0].y;
        if (isCollision) {
            return true;
        }
    }
}
function isWallCollision() {
    const isLeftWallCollision = snake[0].x < 0;
    const isRightWallCollision = snake[0].x > canvas.width - 10;
```

```
    const isTopWallCollision = snake[0].y < 0;
    const isBottomWallCollision = snake[0].y > canvas.height - 10;
    return isLeftWallCollision ||
        isRightWallCollision ||
        isTopWallCollision ||
        isBottomWallCollision;
}
function clearGameCanvas() {
    gameCanvas.fillStyle = CANVAS_BACKGROUND;
    gameCanvas.strokestyle = CANVAS_BORDER_COLOR;
    gameCanvas.fillRect(0, 0, canvas.width, canvas.height);
    gameCanvas.strokeRect(0, 0, canvas.width, canvas.height);
}
function createApple() {
    appleXdir = randomTen(0, canvas.width - 10);
    appleYdir = randomTen(0, canvas.height - 10);
    snake.forEach(function checkIsOnSnake(part) {
        if (part.x === appleXdir &&
            part.y === appleYdir) {
            createApple();
        }
    });
}
function randomTen(min, max) {
    return Math.round((Math.random() * (max - min) + min) / 10) * 10;
}
function drawApple() {
```

```
    gameCanvas.fillStyle = APPLE_COLOR;
    gameCanvas.strokestyle = APPLE_BORDER_COLOR;
    gameCanvas.fillRect(appleXdir, appleYdir, 10, 10);
    gameCanvas.strokeRect(appleXdir, appleYdir, 10, 10);
}
function moveSnake() {
    const movedHead = {x: snake[0].x + xdir, y: snake[0].y + ydir};
    snake.unshift(movedHead);
    const didEatApple =
        snake[0].x === appleXdir &&
        snake[0].y === appleYdir;
    if (didEatApple) {
        createApple();
    } else {
        snake.pop();
    }
}
function drawSnake() {
    snake.forEach(drawSnakePart)
}
function drawSnakePart(snakePart) {
    gameCanvas.fillStyle = SNAKE_COLOR;
    gameCanvas.strokestyle = SNAKE_BORDER_COLOR;
    gameCanvas.fillRect(snakePart.x, snakePart.y, 10, 10);
    gameCanvas.strokeRect(snakePart.x, snakePart.y, 10, 10);
}
function changeSnakeDirection(event) {
```

```
        const keyPress = event.keyCode;
        const ARROW_LEFT = 37;
        const ARROW_RIGHT = 39;
        const ARROW_UP = 38;
        const ARROW_DOWN = 40;
        const movingLeft = xdir === -10;
        const movingRight = xdir === 10;
        const movingUp = ydir === -10;
        const movingDown = ydir === 10;
        if (keyPress === ARROW_LEFT && !movingRight) {
            xdir = -10;
            ydir = 0;
        } else if (keyPress === ARROW_UP && !movingDown) {
            xdir = 0;
            ydir = -10;
        } else if (keyPress === ARROW_RIGHT && !movingLeft) {
            xdir = 10;
            ydir = 0;
        } else if (keyPress === ARROW_DOWN && !movingUp) {
            xdir = 0;
            ydir = 10;
        }
    }
</script>
</body>
</html>
```

gameover.js

```
<%@ page contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8" %>
<!DOCTYPE html PUBLIC
"-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd" >
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=ISO-8859-1">
    <style type="">
        <%@include file="/style.css" %>
    </style>
    <title>Игра окончена</title>
</head>
<body>
<h1>Игра окончена</h1>
<div class="canvas-like-div">
    <div class="centered-group">
        <p class="gameover-p">Игра окончена</p>
        <p class="gameover-p">Длина змейки: <%= request.getAttribute("score") %>
        </p>
        <p class="gameover-returning-p">Нажмите Enter, чтобы сыграть ещё раз</p>
        <p class="gameover-returning-p">Нажмите ←, чтобы перейти в меню</p>
        <script type="">
            document.addEventListener("keydown", returnToPage);
            function returnToPage(event) {
```

```
            const keyPress = event.keyCode;
            const ARROW_LEFT = 37;
            const ENTER = 13;
            let form = document.createElement('form');
            form.method = 'GET';
            form.innerHTML = '<input type="hidden">';
            if (keyPress === ARROW_LEFT) {
                form.action = '${pageContext.request.contextPath}/menu';
                document.body.append(form);
                form.submit();
            } else if (keyPress === ENTER) {
                form.action = '${pageContext.request.contextPath}/game';
                document.body.append(form);
                form.submit();
            }
        }
    </script>
  </div>
</div>
</body>
</html>
```